(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
27 September 2001 (27.09.2001)

PCT

(10) International Publication Number
WO 01/71502 A1

(51) International Patent Classification[7]: G06F 11/36

(21) International Application Number: PCT/IB00/00338

(22) International Filing Date: 23 March 2000 (23.03.2000)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant *(for all designated States except US)*: **SUN MICROSYSTEMS, INC.** [US/US]; 901 San Antonio Road, Palo Alto, CA 94303 (US).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: **KRUGER, Stephen** [ZA/FR]; 7, rue Vicat, F-38000 Grenoble (FR).

(74) Agent: **PLAÇAIS, Jean-Yves**; Cabinet Netter, 40, rue Vignon, F-75009 Paris (FR).

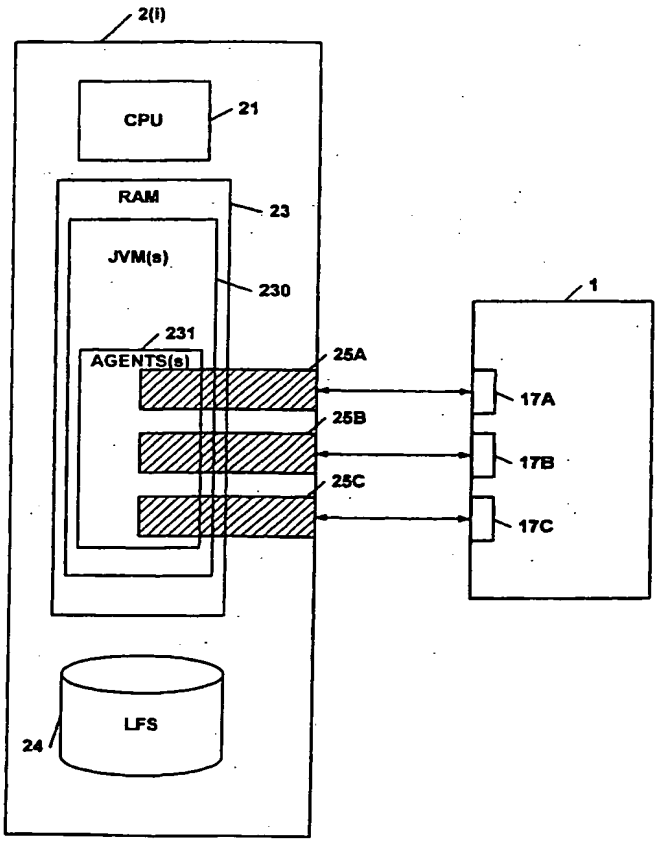
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

[Continued on next page]

(54) Title: METHOD OF AND SYSTEM FOR TESTING SOFTWARE IN COMPUTERS



(57) Abstract: Primary computers (2-i) each have a version of a software to be tested, a local agent (231), capable of having commands to be run in the local agent's primary computer, and interfaces (25A-25C), for communication with a master computer (1). A representation of test operations is stored in the master computer (1). Commands derived from the test operations are dynamically and selectively sent from the master computer (1) to the local agent(s) (231), as designated by a name and a port number. Java virtual machines (230) may be used in the primary computers.

WO 01/71502 A1

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

Method of and system for testing software in computers

This invention relates to computer technology. More particularly, it deals with testing the compatibility of software with various computer architectures.

A new release of a software may comprise a particular version for each computer architecture ("target") to be supported. Each such version has to be tested for full operability within the corresponding target architecture. Generally, this implies a number of operations: firstly, a separate software test program must be written and stored for each type of architecture, and of course for each software version; secondly, these test programs have to be copied by a test operator from a movable medium storage, or across a network connection, into the respective targets provided with the software to be tested; thirdly, the test programs are executed one by one; finally, the result reports or logs have to be recovered from the targets back to the testing facility for common archival and further analysis. This method of testing is thus fairly complex and time-consuming.

In fact, the complexity of this testing process is increased because: a) in general, several successive provisional releases are prepared before the final software release, and b) very often, the software is prepared for several operating systems (including different versions of the same OS), for at least some of the hardware/software combinations.

The testing process complexity is still further increased when the software to be tested is installed in a computer network comprising computer targets with different architectures, and has networking functions, since the operation between each possible combination of targets also has to be tested.

So, it is desirable to reduce the time and the number of

- an instruction for storing data in association with a local agent;
- an instruction for sending at least a portion of the software to be tested to a local agent, for storage in the primary computer.

In an interesting embodiment, which is not limitative, the master computer and the target(s) are equipped with a common environment, preferably defining at least one Java virtual machine in each computer, and the local agent(s) is (are) written for this common environment.

The invention also extends to the software code of a test program in a master test computer, and to the software code of a local agent.

Other alternative features and advantages of the invention will appear in the detailed description below and in the appended drawings, in which :

- figure 1 is a general diagram of a computer network in which the invention is applicable;

- figure 2 diagrammatically shows the organization of a primary computer or target 2 of figure 1;

- figure 3 diagrammatically shows the memory organization of the master computer 1 of figure 1;

- figure 4 is a flow-chart of a local agent in a target 2;

- figure 5 shows the test supporting files required for creating a specific test environment in a master computer;

- figure 6 shows how the successive instructions of a script in the master computer are executed;

- figure 7 is a flow-chart of the execution of a single instruction in the master computer; and

- figure 8 is a flow-chart including the interaction of the master computer with the test personnel.

Making reference to software entities imposes certain conventions in notation. In the detailed description:
- the quote sign " may be used as character string delimiter when deemed necessary for clarity (e.g. "/usr/bin/java"),
- square brackets may be used to frame the optional portion of an entity or name (e.g. "[,VALUE]").

In the following specification, reference will be made to the terms "station" or "host". These terms must be considered as involving a computer. In turn, "computer" means a physical machine, or a virtual machine.

Considering a computer (physical or virtual machine), the expression "software and hardware architecture" refers mainly to the operating system (OS) and to the motherboard, respectively.

The invention may apply to at least one primary computer or target 2-i (i = 1 to N), and to at least one master computer 1 or "harness". The example in the foregoing description considers only one harness computer; however, several harnesses might be used simultaneously as well. Similarly, several targets are considered in the example, although only one target might be used as well. Furthermore, at least in certain situations, the harness(es) and target(s) may be hosted in the same physical machine.

The harness 1 and targets 2-i are preferably interconnected through a network, as illustrated in Fig. 1, where i = 3. However, this networking feature is not absolutely necessary, since e.g. the harness and the agents may be run on the same physical machine. Additionally, both the harness(es) and the target(s) may be locally resident or they may be distributed (e.g. as with the server and client relationship) where communication between the two takes place over some sort of

network connection e.g. LAN, Internet, etc.

The expression "release version" hereinafter means a new release of a software, in a version adapted to a particular computer architecture (and programming language, if appropriate), or more generally any version modified for some reason.

As shown in Fig. 1, harness 1 has a central processing unit or CPU 11, a RAM 13 (or any other working memory, for example DRAM, SDRAM or flash memory), and a hard disk 14 (or any other mass memory, for example floppy disks, removable drives, or flash memory emulating a disk). It may also have input/output devices like a display 15, a keyboard 19, and a pointer e.g. a mouse 18. A graphic user interface (GUI) may also be included if desired. The interconnections within harness 1 are diagrammatically shown as a single bus 10, although a more complex bus structure is generally used.

The architecture of harness 1 is mainly defined by its CPU 11 and its operating system (OS), stored e.g. in hard disk 14. The OS may be, for example, Solaris (a registered trademark of SUN MICROSYSTEMS) or Windows NT (a registered trademark of MICROSOFT).

Harness 1 further has TCP sockets 17-1 through 17-3, for selective communication with corresponding TCP sockets 25-1 through 25-3 of respective targets 2-1 through 2-3 (more generally targets 2-i, with i = 3 on Fig. 1). Other communication interfaces may be used in lieu of TCP sockets.

Each target 2-i (Fig. 2) comprises a CPU 21, a RAM 23 (or any other working memory, for example DRAM, SDRAM or flash memory), a hard disk 24 (or any other mass memory, for example floppy disks, removable drives, or flash memory emulating a disk) and optional peripherals (not shown), which may include dedicated equipment, such as network adapters, removable storage media, or pluggable hardware components. A target OS is stored e.g. in hard disk 24. A local file system

(LFS) may be organized in hard disk 24 ("local" here means visible to the target, but possibly residing elsewhere, as e.g. in a Network File System or NFS scenario). This is exemplary only, and the invention may apply to other hardware target structures.

As shown in Fig. 1, the targets may also comprise communication equipment enabling them to be interconnected in a network via a medium 9, which may be comprised by e.g. Ethernet or token-ring running e.g. on fiber optics or twisted pair cabling or radio transmission or infra-red transmission. Harness 1 may also be connected to the network.

The targets may have various architectures in terms of hardware and/or software. For example, the target architecture may be a SOLARIS-INTEL, a SOLARIS-SPARC, a WINDOWS NT-INTEL, a WINDOWS NT-ALPHA or a WINDOWS 95-INTEL one. (SOLARIS, SPARC are registered trademarks of SUN MICROSYSTEMS; INTEL is a registered trademark of INTEL Corp.; WINDOWS 95 and WINDOWS NT are registered trademarks of MICROSOFT; ALPHA is a registered trademark of DIGITAL EQUIPMENT). The targets may range e.g. from notebooks to desktop stations.

In the example shown in Fig. 2, the target 2 has in RAM a Java virtual machine or JVM 230, which in turn hosts a local agent 231. The local agent 231 has a TCP server socket 25 to accept connection from the harness, as necessary. These connections are created as requested by the harness, e.g. as illustrated in 25A-17A to 25C-17C.

It is preferable, although not necessary, that the operating systems of both the harness 1 and the targets 2-i be equipped with a common environment, which the local agents are written for. In this particular example, the Java environment is discussed. However, the invention is not so limited. Thus, as shown in Fig. 3, the hard disk 14 of harness 1 contains OS files 140, Java supporting files 141, and files 8 supporting the software tests. The Java files 141 are used to implement

a Java virtual machine 130 which is executed in the RAM 13 of machine 1, and hosts a "logical" harness 131, which is in fact a program running the test for the target(s) remotely, from the harness, using test supporting files 8, and sockets 17-i.

In each target, the local agent 231 acts as a kind of "mini-server", capable of communicating with harness 1 on request from this harness, and of running commands in its hosting target 2. Concretely, the local agent is a program written in a language adapted to the target architecture, and stored in its memory 23. The language may be Java when the target implements a Java virtual Machine. Alternatively, other languages may be used, according to the target architecture and environment.

This local agent program 231 (more briefly "local agent") is arranged for managing the local target TCP socket 25. This enables communication between the target local agent 231 and the harness 1, independently of other target local agents. So, each local agent is attached to one TCP socket 25 of its hosting target computer 2. As illustrated in Fig. 2, if three local agents are implemented in the same target computer, that target computer will have three TCP sockets 25A through 25C, each of which may be implemented through a server socket. Although each local agent is preferably hosted in its own JVM, this is not shown in the simplified representation of Fig. 2.

The basic operation of a local agent is shown in the flow chart of Fig. 4. Normally (step 40), a local agent waits for receipt of a request from the harness, on its dedicated TCP socket. Upon such a receipt, step 42 may optionally check e.g. the syntax and/or arguments of the request and its arguments, at least partially. If the request is found not consistent (at 44), then an error is returned to the harness at step 45, and control is given to step 40. Otherwise (normally), step 47 launches execution of the request in the local target ; a report, together with results if

appropriate, is returned to the harness at step 49, and control is returned to step 40 of waiting for another request from the harness.

The program for such a harness and local agent interaction may be written e.g. in Java, or in any other desired language, including Interface Definition Language (IDL) of the Object Management Group (OMG), and converted into an executable version in the target. A number of functionalities may have to be implemented in the local agent, depending upon the set of harness commands to be executed in the target, as described hereinafter. Programming each of these functionalities is considered accessible to men skilled in the art.

For purposes of software testing, each target 2-i is provided with a copy, or a portion of a copy, of the software to be tested, in a release version matching its own architecture (as above defined), and also matching its supported programming languages, if appropriate.

The release version may be previously installed in the mass memory 24 of the target, or selectively sent from the harness 1 to the target 2-i e.g. through the network, just before running a test. The release version may be an application, which may involve information obtained through communication with other computers, or from local files stored in the mass memory or hard disk 24.

Harness 1 is used for selectively distributing specific test sequences to the targets 2-i, in which respective release versions have to be tested. As mentioned before, the specific sequences are basically intended for selecting the release versions to correctly operate within the target architectures.

For this purpose, harness 1 has test supporting files 8 stored in its hard disk 14. In the example of Fig. 5, the test supporting files 8 firstly comprise agent definitions

81, which comprise agent variables (or agent properties, for e.g. as given in Example A.1 in Exhibit A). For a given local agent hosted in a given target, the agent properties include at least a portion of the information needed at the harness level for sending that local agent a request being executable in the target.

The test supporting files 8 further comprise a generic test section 84, having, in the example:
- target OS specific test code, e.g. Solaris specific test code 841S and NT specific test code 841N;
- optionally, versions of the software to be tested for various target architectures, e.g. a Solaris version 842S and an NT version 842N; and
- generic test code 850, examples of which will be given hereinafter.

The test supporting files 8 may further comprise optional default settings 82, corresponding e.g. to standard system directories as used with Solaris or NT.

Seen from harness 1, each local agent is preferably identified by a unique couple or doublet of parameters, comprising a name which identifies the local agent (and, hence, its host target), and a port number which identifies its TCP socket. While this is an Internet Protocol (IP) type identification, other non IP identifications might be used as well. This couple may be the only information known by the harness about the local agent, i.e. all the other information about the local agent will be determined at the time of executing the test, by querying each target for more detailed information through appropriate usage of the test script code, as illustrated in an example on Figure 6.

Generally, a local agent 231 will execute commands sent from harness 1. Such a command may result in an action in the target computer hosting the local agent; the action may be internal, or comprise sending file data required by harness 1, or sending file data required (on command) by another

local agent, to be transmitted to harness 1; data representing the test result in the target computer 2 hosting the local agent may also be returned to harness 1.

In accordance with this invention, a representation of a sequence of test operations is stored in harness 1. Then harness 1 will selectively send commands derived from at least certain of the test operations, to a local agent. The selection of commands may be based on available information about the target agent. At least part of the available information may be dynamically acquired.

When sent to a given local agent, the sequence of test operations has to be configured in accordance with the specific environment of the target 2-i hosting that given local agent, including its operating system (OS) and hardware architecture.

In accordance with another aspect of this invention (Fig. 6), the sequences of test operations are loaded in RAM 13, e.g. from one or more files. This forms a script 60, which is read by a harness test parser 64 also loaded in RAM 13, taking into account the current designated local agent (agent name and agent port number i.e. the agent doublet of parameters). Fig. 6 also shows auxiliary data files 62, for use in implementing the script, and stored in hard disk 14. The parser 64 accesses the instructions in the script 60 one by one. The result of the parsing, at 66, is a sequence of one or more commands, which are now in an executable form. The number of commands in the sequence depends upon the design of the basic script file 60, and upon the interaction of the user with parser 64. The commands are then executed at 70.

With reference to Fig. 7, the execution 70 starts with a first command as the current command, which is executed at 71. If appropriate (depending upon whether the command is at least partially executable on the target), the command is sent as a request to the local agent being involved. If the command is not successfully executed (72), an error

processing is made at 73. Otherwise (normally), if step 74 indicates further commands, step 75 goes to the next command, and control is given to step 71. Step 79 is the end of the processing of the sequence of commands.

Thus, a test sequence module, comprising a representation of a sequence of test operations, is formed on the basis of the test supporting files 8, used as desired by the script 60. The parser 64, its products 66 (a working area in RAM), and the execution module 70 together act as a test control module, based on the script 60.

Preferably, instead of being written in a form being directly executable in the target, the instructions in the test sequences refer to parameters (or variables), which are defined later. Some of these variables may be file names, thus enabling to select a file depending upon the target architecture. This is made preferably by using a "master language" (a language adapted to control the test from the harness) to write these instructions ("master instructions"). The instructions in the master language are e.g. interpreted at the time of their execution, i.e. at step 70 latest.

Non limitative examples of master language instructions will now be given.

GETFILE(AGENT_ID,SRC_PATH,DEST_PATH) is a request instruction for retrieving an arbitrary file from a local agent designated by AGENT_ID, in its path SRC_PATH, and sending it to path DEST_PATH, relative to the harness 1.

SENDFILE(AGENT_ID,SRC_PATH,DEST_PATH) is a request instruction for sending an arbitrary file to a local agent AGENT_ID, from a path SRC_PATH, relative to the harness 1, to a path DEST_PATH relative to the target having the local agent designated by AGENT_ID.

In case of local agents running on various targets 2-i, it is advantageous to use target independent methods for specifying

the paths. The following examples are merely exemplary.

SETAGENT(AGENT_ID,AGENT_NAME,AGENT_PORT,WORK_DIR) is a master instruction which sets up a local agent for use during the lifetime of a current master instruction file. In the arguments:
* AGENT_ID (return parameter) designates the local agent;
* AGENT_NAME designates the Java virtual machine running the designated local agent;
* AGENT_PORT designates the port number of the designated local agent;
* WORK_DIR designates a directory to place any files sent to the designated local agent via a ZIPSEND command (defined hereafter).

ZIPSEND(AGENT_ID,ZIP_ID,ZIP_FILENAME) is a master instruction, which may be defined with three arguments:
* AGENT_ID designates the local agent to send the zip file to (the local agent has been created by a previous call to SETAGENT);
* ZIP_ID (return argument only) designates the handle to the zip file who will be created. So, it must be undefined at this point and will be used thereafter when a call is made to a ZIPDEL command (delete a previously sent ZIPFILE);
* ZIP_FILENAME designates the full path to the zip file to be sent, relative to the harness 1.

ZIPSEND will send the zip file ZIP_FILENAME from the harness to a local agent designated in AGENT_ID and request that agent to unzip the file. The unzipped file is placed in the directory designated by WORK_DIR in the SETAGENT instruction for that AGENT_ID.

SETPROP(AGENT_ID,KEY[,VALUE]) is a master instruction which sets a property value on a specified AGENT_ID, which will be visible to any subsequent CMDSTART commands (defined hereafter) called on this AGENT_ID. In the arguments:
* AGENT_ID which designates the local agent on which the property must be set;

* KEY which designates the key name of the property;
* VALUE (optional third argument) designates the value of the property. If this argument is omitted, it will be retrieved from the specific chosen syntax environment, and if it is not defined here either, it will be assigned to the same value as KEY.

For example, SETPROP(myAgent,"System Root","c:\winnt") sets the "System Root" property of local agent "myAgent" to the value "c:\winnt".

INCLUDE(OTHER_FILE) is a master instruction used to include another master instruction file OTHER_FILE into the current file.

CMDSTART(AGENT_ID,CMD_ID,CMD_STRING) is a request instruction which spawns an execute process on a local agent, using an environment built from any previous calls to SETPROP, and then executing the command specified in CMD_STRING (defined hereafter). It accepts at least three arguments:
* AGENT_ID which designates the local agent on which the command is to be executed;
* CMD_ID which designates a handle to be used for further communication of the started process;
* CMD_STRING which designates the command to be executed on the designated local agent. It must match the OS and hardware architecture of the target hosting the local agent.

REPORTSTATUS(STATUS_CODE) is a request instruction used to ask for a report of the final value of a test to the harness 1. It serves to decide whether a test passed or failed. For example, a value of 0 will indicate a test passed, any other value indicating an error (e.g. failed, not run, or unresolved). Accordingly, in case of multiple test processes on multiple agents, each partial result can be combined by a syntax file and then a test result can be decided upon. It accepts a single return-only argument:
* STATUS_CODE which designates any integer value, one of those signifying a test passed.

The master instructions may include other self explanatory instructions, like DELFILE, or CHECKFILE, and so on. The syntax files for the master language and the system local files are preferably stored in hard disk 14 of harness 1. Thus, it is possible to transfer files, unzip files, delete files, set specific environment parameters (or variables), and more generally do anything needed for running a test, in a target 2-i, under control of the harness 1. Moreover, with these master instructions, once a test sequence has been run, a result is decided on, and may be sent back to the harness 1 for processing and, if desired, for storage in hard disk 14 (Fig. 3).

The master instructions are used by the test parser for building up dynamically the desired environment (or test instruction execution configuration) adapted to data communication between the harness 1 and each designated target local agent. This dynamic test creation is made and managed by test control module 6.

Thus, when writing the test, one does not need to know a priori what type of machine the test will run on.

Moreover, the file transfer facility (SENDFILE) allows the release version which is to be tested, to be sent to the local agent machine 2-i as well, before executing the test. Thus, one no longer needs to manually re-install the local agent machine 6 with each new release version. Furthermore, since the local agent-hosting target communication is connectionless, even rebooting the target computer is allowed, as long as the local agent automatically restarts during the boot-up process.

A non limiting example of a set of master instructions is shown as Example I in Exhibit A. It will send a file "passtest.zip" to a local agent, unzip it, run the Java class contained in this zip file "PassTest", and report the result to the test harness. Lines beginning with "#" are not executable.

It is assumed that a local agent "AGENT1" has been defined from an agent definition file 81. The agent definition includes a variable AGENT1_OSNAME, containing the name of the OS in the corresponding target. The value of AGENT1_OSNAME may be "Windows NT" or "Solaris".

Section A.1 defines default hard-coded settings: the path to the Java home directory in the target; the path and name of the Java Interpretor in the target; a keyword representing the target OS class; the path to a temporary directory in the target; and a name TEST_NAME given to the current test sequence. In section A.1, the variable "FILE_SEPARATOR" is referred to the agent machine, seen from the harness. Java supplies another variable noted "file.separator", for the local machine (whether harness or target). The default settings might be imported from one of the files 82, for example by using the instruction INCLUDE(OTHER_FILE) in lieu of section A.1.

Section A.2. sets up an agent to run the test on. "AGENT1_NAME" and "AGENT1_PORT" are variables previously defined by the user, via the harness interface. "AgentID" is returned after execution of the instruction.

Section A.3 fetches a zip file "passtest.zip" containing the test, in a subdirectory of the harness, and sends it to the local agent in the target. "Passtest.zip" is one of the files labelled 62 in Fig. 6.

Section A.4 executes the test, using a timeout of 30 seconds. More precisely, it launches the Java virtual machine to execute the Java class "PassTest" unzipped from "passtest.zip" in the target temporary directory named "TMP_DIR".

Section A.5 gets the status returned by the execute command. Section A.6 gets the standard output and standard error traces. Section A.7 cleans the standard output and standard error traces. Section A.8 cleans the zip file on the agent

side. Section A.9 clears any leftover agent files.

Finally, section A.10 reports the status to the harness user interface, e.g. a GUI.

The generic test code 850 of Fig. 5 is comprised of sets of instructions which are target independent, like the sets of instructions in Examples III and IV in Exhibit C. These examples are given only to illustrate what target independent code may be, and are not intended to constitue a part of an actual test code.

The above discussed example shows how the sequence of test operations contained in the file "passtest.zip" will be executed in a target. As noted, the hard-coded setting variables for each target may be contained in a target dedicated file, which is simply called before execution of the test sequence itself.

It should be now understood that the test parser is responsible for parsing the script 60, and for adapting the same according to the designated agent name and agent port number (the agent doublet of parameters). In the example of Exhibit A, after an agent is set, that agent is referred to by its AGENT_ID only.

The general structure of a test specification (script) may include the following steps:
- first, each local agent is queried for its operating system (OS) and hardware architecture.
- then, some of the local agents are "SET" for execution of the test. The SETAGENT instruction is valid until a corresponding DELAGENT is executed. Note a DELAGENT simply means that the local agent is no longer seen from the harness resident test program.
- then, using the AGENT_ID as necessary, the harness builds up the test commands which will be sent to each of the local agents involved in the test specification, for the test to be run. Each sent command has its own "CommandID".

- at the end, the results are stored in 89 (Fig. 5).

Accordingly, the script of master instructions may generally comprise any of the following, in any desired order (compatible with parsing requirements):
- set initial condition statements
- set variables
- set properties
- send commands to local agents
- retrieve command results,
- decide whether the unitary test (set of master instructions) passed or failed.

The script is conveniently handled using a parser and editor based on a Graphic User Interface or GUI. It is also possible to run a test sequence without any GUI interface, if one makes use of a parser command-line interface, for example while specifying :
* a session name which indicates the number of the run, which will be used to store the results retrieved from the local agent(s) in a dedicated home directory;
* a property file which contains all the variables (or parameters) required to parse the test operations to be run;
* a test path which specifies the full path to each syntax file to be run.

Reverting to Fig. 5, hard disk 14 stores a collection of test specifications 841, corresponding to different levels, including:
- OS level, e.g. Solaris or NT,
- level of languages supporting the software to be tested, like Java 1.1 or Java 1.2 in the Java implementation,
- level of the hardware architectures, like Intel X86, Alpha, or Sparc.

These various levels may be analyzed in a single test script by constructing a tree like optional structure, using e.g. IF statements, and sequentially considering the pertinent options in the above indicated levels. This is illustrated by

Example II in Exhibit B. Actions may be taken within the tree, or when reaching a leaf of the tree. Actions may comprise a CMDSTART using an adequate one of the command files 841.

The function of the test script (and of its master language) is to elucidate, based on the initial information known about each target agent (host and port number), enough information to send to the agent all information and/or data required to properly exercise the software being tested. This information is sequentially built up by dynamically querying the agent at the time of executing the test, and/or building on previously queried information and/or supplied default values (default properties) for each of the tested architecture/OS combinations.

When parsing such a script, the test parser considers the local agent being set therein, and progressively builds up a test configuration adapted to that local agent. If several local agents are set, they are processed selectively the same way, with each agent finally receiving test commands adapted to its own architecture.

After the parsing of all or part of the script, the logical harness 131 forms, in RAM 14, a set of agent specific commands, which are ready for being directly sent as requests to the local agent of a target 2-i, via sockets 17-i and 25-i.

The above mentioned collection of test specifications 841 may be managed as a database, where each test specification is associated to its corresponding OS, supporting language, and hardware requirements. It may also include an identification of the corresponding specific software release version to be tested.

Thus, in the script, most or all of the test operations themselves may be written in an agent-independent manner, with a syntax matching the harness test parser.

The "SETAGENT" portion of the script, and the corresponding analysis of "levels" may be considered as establishing a correspondence between active local agents and records in the above mentioned database. This correspondence may be materialized by a correspondence table between local agents (whose identifiers are defined) and corresponding test specifications (sequences of test operations) to be run with a release version in targets 2-i.

For building up the test configurations, the harness 1 may also comprise a configuration file for each local agent registered in the correspondence table (or, at least, a configuration file for each local agent involved in the currently processed test sequences).

Generally, the test sequences in the harness include master instructions, the execution mode of which is stored in mass memory 4. The master instructions may have one or more arguments (or parameters). In fact, most of the master instructions are request instructions, i.e. will result into a request being sent by the harness to one of the local agents; again, in most cases, one argument (or more) in the request instruction is a command to be executed in the target, or includes such a command.

In case of Java language, all that is required to run any test sequence on any target computer is a Java virtual machine running a local agent in that target computer. However, the invention may be correspondingly extended for implementation using other languages.

Thus, to execute a sequence of test operations, residing on the harness 1, one just needs to configure the target specific elements of each supported agent machine architecture. The configuration values typically do not change after the initial set-up of the system.

Moreover, the harness 1 may allow the user to effect permutations of the local agents, and since the local agents

can be queried, it is possible to detect, just prior to running, what the destination set-up (Java virtual machine, O.S., hardware architecture) needs to be, and to send the corresponding commands (and requests) comprising the test sequence to the local agent(s).

It is now assumed that a current test combination is running, testing for example networking operations in a NT-Solaris environment (a target uses NT, another one Solaris). Switching to a Solaris-Solaris environment involves designating the local agents one wants to run the networking test on; everything else may be automatic.

Men skilled in the art will understand that the invention applies independently of the particular "master language" used to write the instructions of the test sequence. The harness 1 may allow users to define their own shell script parser for use with the other testing tools.

The parser in the given implementation of the technology is implemented as a Java class interface to allow abstraction of the actual parser details from the GUI, and to facilitate the implementation of other types of parser by simply implementing the provided interface "ParserInterface", shown in Exhibit D. Men skilled in the art will be able to implement their own master language syntax using this standard Java abstraction mechanism.

If the technology was implemented in a language other than Java, a similar abstraction method would be used, such as a Corba IDL interface, or include file header.

Whether the harness may run on Windows NT or Solaris or any other hardware/software configuration (or several of them) depends on how portably the test sequences are written, as far as the master computer architecture is concerned ("portably" refers here e.g. to the master instructions references to the auxiliary data files 62, and other platform specific properties required to test the software, such as

absolute file paths, drive letters, interrupts, etc).

The approach of this invention greatly facilitates the distribution of tests, by centralising them in a single master computer (harness) and dynamically selcting and distributing these tests, when needed, to simple target local agents, which act as "mini-servers" waiting for commands to be run on the "target" primary computer provided with the software to be tested.

Thus, a kind of test database, associated with a number of different target computer architectures, is stored in the harness computer. Accordingly, a test operator will simply have to designate, in the database, one (or more) local agent(s) intended to run a selected test; after that, a specific test environment is dynamically created between the harness computer(s) and the designated local agent(s) in their respective hosting target computers.

In other words, effectively running a software test sequence now consists in affecting "values" to the parameters (or variables) identifying every target local agent involved in that particular software test sequence.

This database may be modified or completed with new test sequences every time a new computer architecture appears.

The test system may be built as follows:

- there is provided for primary computers 2-i having respective specific architecture (in terms of software and hardware). Each primary computer 2-i is provided with a software (or application) to be tested and at least one local agent. This software and the local agent(s) are adapted to their host primary computer architecture, and each local agent is capable of communicating data and running the software, on command.

- provision is also made for a master computer or harness 1,

which may store a correspondence between local agents and sequences of test operations to be run with a software to be tested in a primary computer 2-i. The or each master computer 1 may also communicate with the primary computer local agent(s).

In operation (Figure 8), the test personnel will:
- designate at least one primary computer local agent, e.g. in a table (step 101). This may be implemented with a GUI. The designation consists preferably of a couplet of parameters (or variables) such as a name identifying the local agent primary computer and a port number identifying the TCP socket 5 attached to the local agent.
- select each sequence associated with each designated primary computer local agent (step 103);
- at step 105, the master computer builds up dynamically a specific test environment (or configuration) adapted to data communication between the master computer(s) 1 and each designated primary computer local agent. This may be done with a parser.
- as a result (step 107), data representative of the selected sequence of test operations are communicated from the master computer 1 to the selected primary computer local agent(s), in the specific test environment, for running the desired test operations with the software to be tested, in the primary computer(s) 2-i.
- at step 109, the harness gathers the results, i.e. reports sent back from the primary computer.

The invention is not limited to the embodiments and processes described above, which are given solely by way of example; many alternative embodiments may be envisaged by men skilled in the art.

For example, multiple local agents may reside in the same target, or may be spread across several targets, as long as all the doublets designating the local agents are unique. This may be of use for example when the release version to be tested is an application requiring results or information

provided by several targets.

On another hand, the invention is not limited to local agents and/or harnesses written for implementation in a Java virtual machine.

This invention also covers the proposed software code itself, especially when made available on any appropriate computer-readable medium. The expression "computer-readable medium" includes a storage medium such as magnetic or optic, as well as a transmission medium such as a digital or analog signal. The software code basically includes the code for use in the harness, and the code for use in the targets. It may also include the accompanying files, and further include at least some of the group of software components to be installed.

The software code of a test program in the master test computer comprises:
- code adapted for selective communication with a remote agent,
- code forming a test sequence module comprising a representation of a sequence of test operations, and
- code forming a test control module adapted for sending commands derived from at least certain of the test operations, from said master computer to a selected local agent.

The software code of a local agent comprises code capable of having commands to be run in an hosting computer, and capable of communication for receiving such commands.

EXHIBIT A - Example I

```
# A.1
JAVAHOME_NT="c:\jdk1.2.2"
JAVAHOME_SOLARIS="/usr/java1.2"

IF (AGENT1_OSNAME == "Windows NT")
 THEN
   FILE_SEPARATOR = "\"
   JAVA_INTERPRETOR=JAVAHOME_NT+FILE_SEPARATOR+"bin" +
FILE_SEPARATOR+"java.exe"
   KEYWORDS = "windows"
   TMP_DIR = "c:\tmp"
 ELSE
   FILE_SEPARATOR="/"
   JAVA_INTERPRETOR=JAVAHOME_SOLARIS+FILE_SEPARATOR+"bin" +
FILE_SEPARATOR+"java"
   KEYWORDS = "solaris"
   TMP_DIR = "/tmp"
END
TEST_NAME = "Test pass example";

# A.2.
SETAGENT(AgentID, AGENT1_NAME, AGENT1_PORT, TMP_DIR);

# A.3
ZIPSEND(AgentID, zipID,
TESTPATH+file.separator+"tests"+file.separator+"zips"+file.separator+"passtest.zip");

# A.4
CMDSTRING = JAVA_INTERPRETOR+" -classpath "+TMP_DIR+" PassTest";
CMDSTART(AgentID, CommandID, CMDSTRING, "30");

# A.5
CMDSTATUS(CommandID, StatusCode);

# A.6
CMDGETTRACE(CommandID);
```

```
# A.7
CMDCLEAN(CommandID);

# A.8
ZIPCLEAN(zipID);

# A.9
DELAGENT(AgentID);

# A.10
REPORTSTATUS(StatusCode);
```

EXHIBIT B - Example II

```
# OS level
IF (AGENT1_OSNAME == "Windows NT")
 ... set corresponding variables
 MY_LEVEL = "NT"
END
 .... (other cases of OS)

# Supporting language level
IF (TEST_SUPPORT == "Java 1.1")
 ... set corresponding variables
 MY_LEVEL =  MY_LEVEL + "J11"
END
 .... (other cases of language)

# hardware level
IF (AGENT1_HARDNAME == "Intel X86")
 ... set corresponding variables
 MY_LEVEL =  MY_LEVEL + "X86"
END
 .... (other cases of hardware level)

# leaf level (final selection of test)
IF (MY_LEVEL == "NTJ11X86")
# do the same as A.4 in Example 1. This includes the CMDSTART in A.4,
# applied to a command corresponding to that value of MY_LEVEL.
# if necessary, A.4 is preceded by A.3, to send the file corresponding
# to that value of MY_LEVEL to the target.
# steps A.5 through A.10 may follow, as necessary.
END
 .... (other values of MY_LEVEL)
```

EXHIBIT C

Example III

```
# Add up all odd numbers between 0 and 10
ODD_NUMBER_SUM = 0
FOR I = 1 TO 10 BY 2 DO
   ODD_NUMBER_SUM += I
END
PRINT "Sum of odd numbers between 1 and 10 is " + ODD_NUMBER_SUM
```

Which will generate the output :

Sum of odd numbers between 1 and 10 is 11

Example IV:

```
# Generate a random number
RANDOM(RANDOM_NUMBER)
PRINT "We generated the random number :"+RANDOM_NUMBER
```

Which may generate the output :

We generated the random number :54324554

Other more sophisticated examples of such code may be generation of Fibonacci sequences, prime number generation.

EXHIBIT D - Methods of the PARSERINTERFACE class

getSyntaxKeyWords

public java.lang.String[] **getSyntaxKeyWords()**

This method lists all the keywords supported by this syntax, for use in the Notepad syntax highlighting. It must be declared static.

getSyntaxSeparators

public java.lang.String[] **getSyntaxSeparators()**

This method lists all the separators supported by this syntax, for use in the Notepad syntax highlighting. It must be declared static.

setTestPath

public void **setTestPath(java.lang.String testFile)**

This method sets the path to file file containing the syntax which will be parsed.

setPrintStream

public void **setPrintStream(java.io.PrintStream printStream)**

This method sets the PrintStream to use for reporting errors and other types of output from the script.

setProperties

public void **setProperties(java.util.Properties p)**

This method sets any default properties which will be required for parsing this file.

getProperties

public java.util.Properties **getProperties()**

This method returns all the properties obtained by parsing this test file.

setEvaluationMode

public void **setEvaluationMode(boolean evalMode)**

If set to true, the parser does not actually make contact with the agents but merely simulates the agent responses to allow standalone parsing.

parseFile

public int **parseFile()**

throws java.lang.Exception

This method parses the specified file. If not in evaluation mode, it should return the status of the test run : ProtocolConstants.PASSED ProtocolConstants.FAILED ProtocolConstants.NOTRUN

ProtocolConstants.UNRESOLVED

interrupt

public void **interrupt()**

This method is responsible for killing any processes already started on the agents, and immediately halt parsing any files.

getProperty

public java.lang.String **getProperty(java.lang.String key)**

This method retrieves the specified property from the results of parsing this file.

getProperty

public java.lang.String **getProperty(java.lang.String key, java.lang.String defaultValue)**

This method retrieves the specified property from the results of parsing this file. If the value is not found, the defaultValue is returned.

getTestName

public java.lang.String **getTestName()**

This method should return a test name which will be used to display the test in the test tree.

getKeyWords

public java.lang.String[] **getKeyWords()**

This method should return all keywords associated with this test. These will be used in using the keywords to select/deselect tests in the harness.

getIncludeList

public java.lang.String[] **getIncludeList()**

This should return the list of files other than standard java.util.Properties files which were included to parse this test file.

getPropertiesIncludeList

public java.lang.String[] **getPropertiesIncludeList()**

This should return the list of standard java.util.Properties files which were included to parse this test file.

getTraceList

public java.lang.String[] **getTraceList()**

This method should list all available output files produced by this test when run on the agent, but relative to the harness.

CLAIMS

1. A method of testing computer software, comprising the steps of :

a) providing a primary computer (2-i) with a software to be tested,

b) providing the primary computer with a local agent (231), the local agent being capable of having commands to be run in the local agent's primary computer, and having a communication interface,

c) providing a master computer (1), arranged for selective communication with the communication interface of the local agent (231), with a representation of test operations being stored in the master computer (1), and

d) dynamically and selectively sending commands derived from at least certain of the test operations, from said master computer (1) to the local agent (231).

2. The method of claim 1, wherein, in said master computer, the representation of test operations comprises data stored in correspondence with a designation of the local agent, and a sequence of instructions, at least some of which have the designation of the local agent as a parameter,

and step d) comprises:

d1) scanning the sequence of instructions,

d2) for at least some of the instructions, building a test command from that instruction and the data corresponding to the local agent parameter of that instruction, and

d3) sending the test command to the local agent.

3. The method of claim 2, wherein the designation of a local agent comprises a name given to the local agent.

4. The method of claim 3, wherein the designation of a local agent comprises a port number assigned to the local agent.

5. The method of claim 1, wherein step a) further comprises providing the primary computer (2-i) with a common software environment, and the local agent of step b) is a program

written for that common environment.

6. The method of claim 5, wherein the common environment comprises at least one Java virtual machine (230,130).

7. The method of claim 2, wherein the instructions are selected from a predefined set of instructions forming a test programming language, the predefined set of instructions comprising an instruction requesting a local agent to start execution of a command.

8. The method of claim 7, wherein the set of instructions further comprises an instruction requesting a local agent to report on the status of execution of a command.

9. The method of claim 7, wherein the set of instructions further comprises instructions for sending files to and retrieving files from a local agent.

10. The method of claim 7, wherein the set of instructions further comprises an instruction for instituting a correspondence between a local agent name, a corresponding port in the master computer, and an agent identifier.

11. The method of claim 7, wherein the set of instructions further comprises an instruction for storing data in correspondence with a local agent.

12. The method of claim 7, wherein the set of instructions further comprises an instruction for sending at least a portion of the software to be tested to a local agent, for storage in the primary computer.

13. A system for testing software in computer, said system comprising :

- a primary computer (2-i), provided with a software to be tested, and with a local agent, the local agent being capable of having commands to be run in the local agent's primary computer, and the local agent having a communication

interface,

- a master computer (1), having an interface (17) adapted for selective communication with the respective local agent interfaces, and having a memory comprising:

. a test sequence module (8) comprising a representation of a sequence of test operations, and

. a test control module (64,66,70) adapted for sending commands derived from at least certain of the test operations, from said master computer (1) to a selected local agent.

14. The system of claim 13, wherein the test sequence module (8) comprises:

. data stored in association with a corresponding local agent, and

. a sequence of instructions, at least some of which have a designation of the local agent as a parameter,

and the test control module comprises:

. code (64) for scanning the sequence of instructions,

. for at least some of the instructions, code (66) for building a test command from the instruction and the data corresponding to the local agent parameter of that instruction, and

. code (70) for sending the test command to the local agent.

15. The system of claim 14, wherein the designation of a local agent comprises a name given to the local agent.

16. The system of claim 14, wherein the designation of a local agent comprises a port number assigned to the local agent.

17. The system of claim 13, wherein the primary computer (2-i) is provided with a common software environment, and the local agent is a program written for that common environment.

18. The system of claim 17, wherein the common environment comprises at least one Java virtual machine (230,130).

19. The system of claim 14, wherein the instructions are selected from a predefined set of instructions forming a test programming language, the predefined set of instructions comprising an instruction requesting a local agent to start execution of a command.

20. The system of claim 19, wherein the set of instructions further comprises an instruction requesting a local agent to report on the status of execution of a command.

21. The system of claim 19, wherein the set of instructions further comprises instructions for sending files to and retrieving files from a local agent.

22. The system of claim 19, wherein the set of instructions further comprises an instruction for instituting a correspondence between a local agent name, a corresponding port in the master computer, and an agent identifier.

23. The system of claim 19, wherein the set of instructions further comprises an instruction for storing data in correspondence with a local agent.

24. The system of claim 13, wherein the communication interface of each local agent comprises at least one TCP socket (25).

25. The software code of a test program in a master test computer, comprising:
- code adapted for dynamic and selective communication with a remote local agent,
- code forming a test sequence module (8) comprising a representation of a sequence of test operations, and
- code forming a test control module (64-66) adapted for sending commands dynamically derived from at least certain of the test operations, from said master computer (1) to a selected local agent.

26. The software code of a local agent in a computer,

comprising code (231) capable of having commands to be run in an hosting computer, and capable of communication with another process for receiving such commands.

5

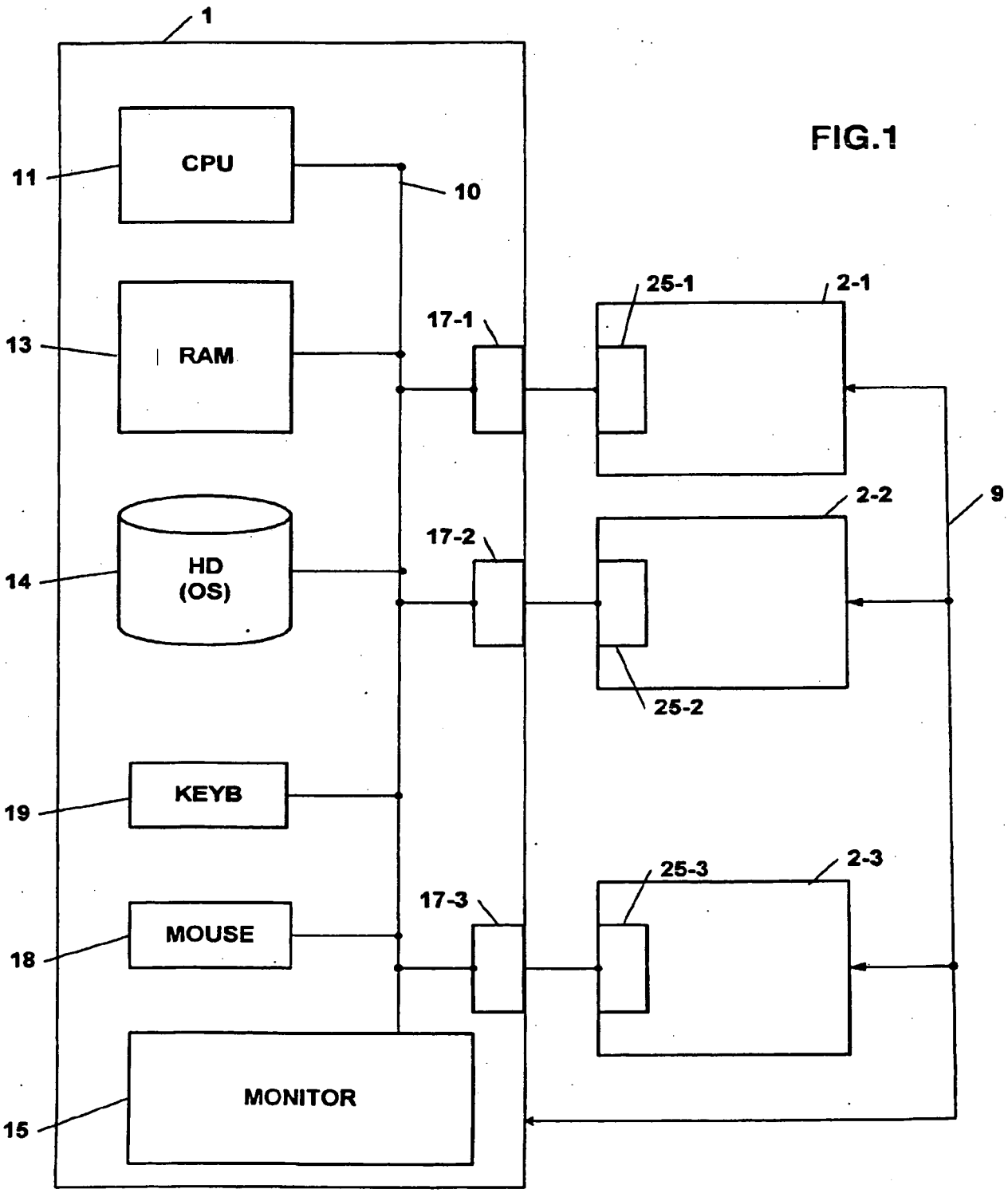

1
CPU
11
10
RAM
13
17-1
25-1
2-1
HD
(OS)
14
17-2
2-2
9
25-2
KEYB
19
MOUSE
18
17-3
25-3
2-3
MONITOR
15

FIG.1

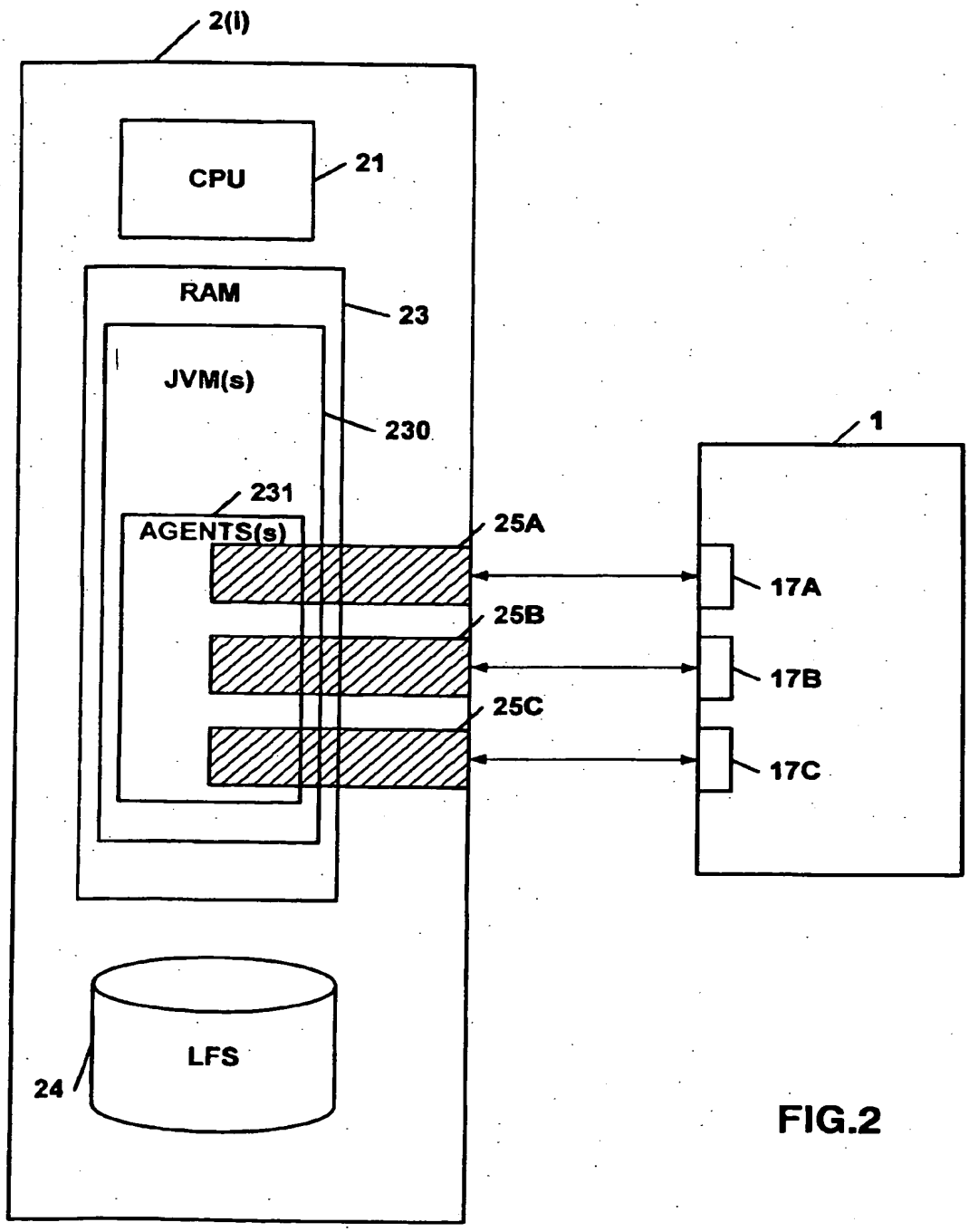

2(i)
CPU
21
RAM
23
JVM(s)
230
231
AGENTS(s)
25A
25B
25C
1
17A
17B
17C
LFS
24

FIG.2

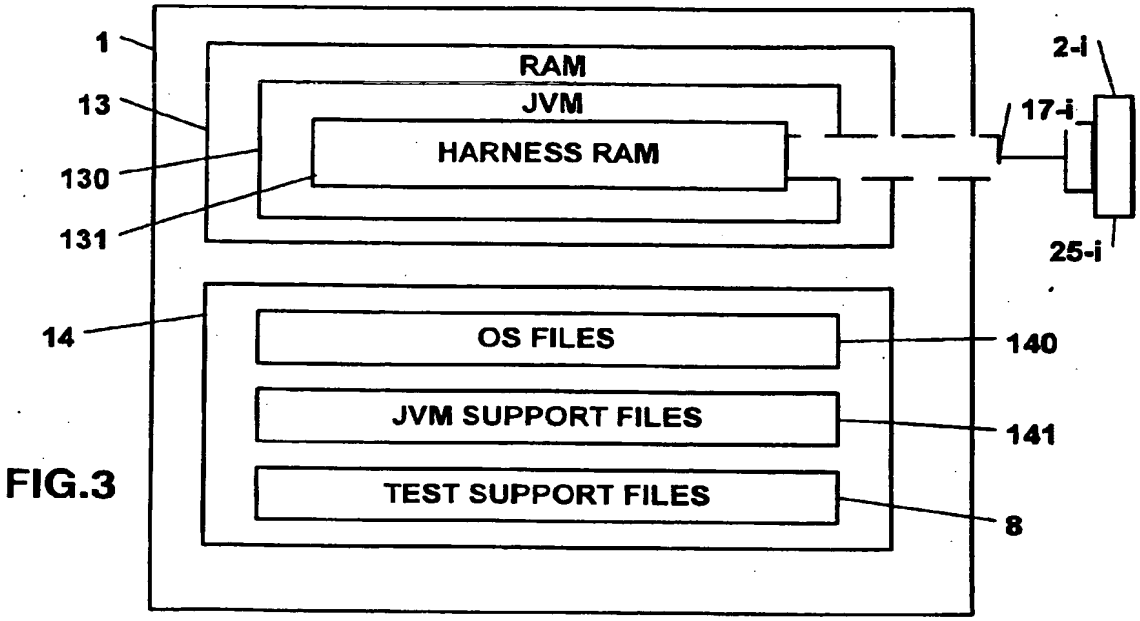

1
RAM
13
JVM
HARNESS RAM
130
131
2-i
17-i
25-i
14
OS FILES
140
JVM SUPPORT FILES
141
TEST SUPPORT FILES
8

FIG.3

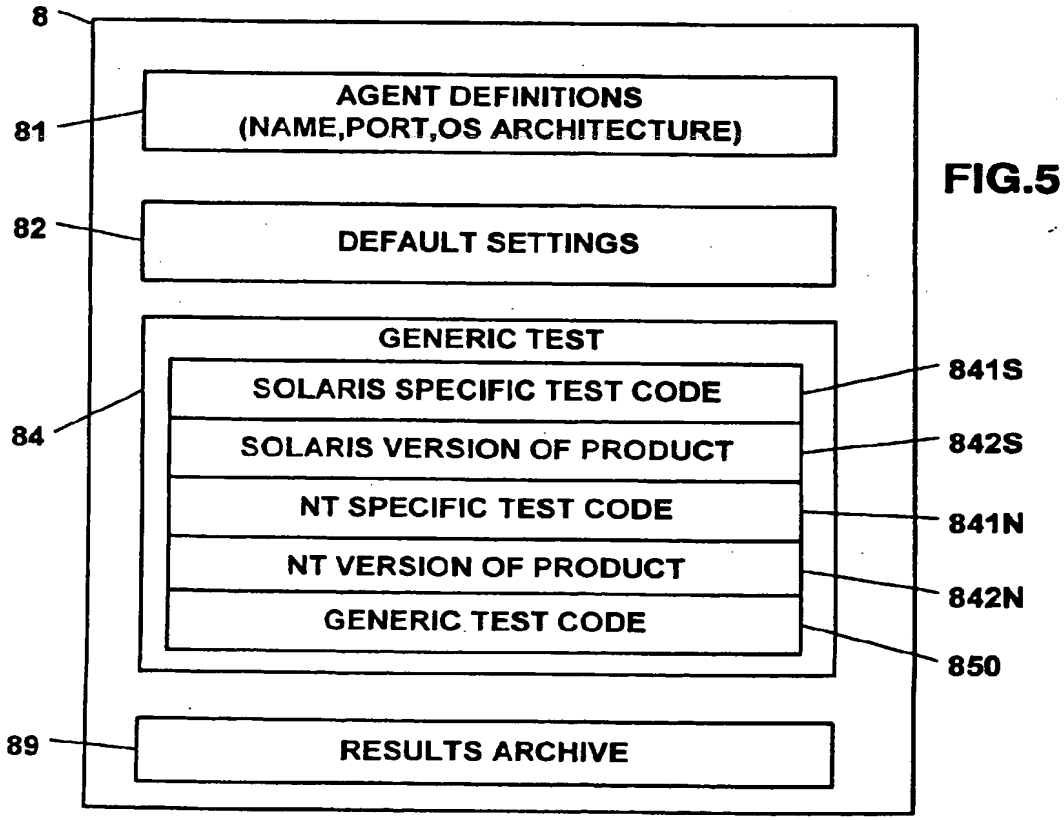
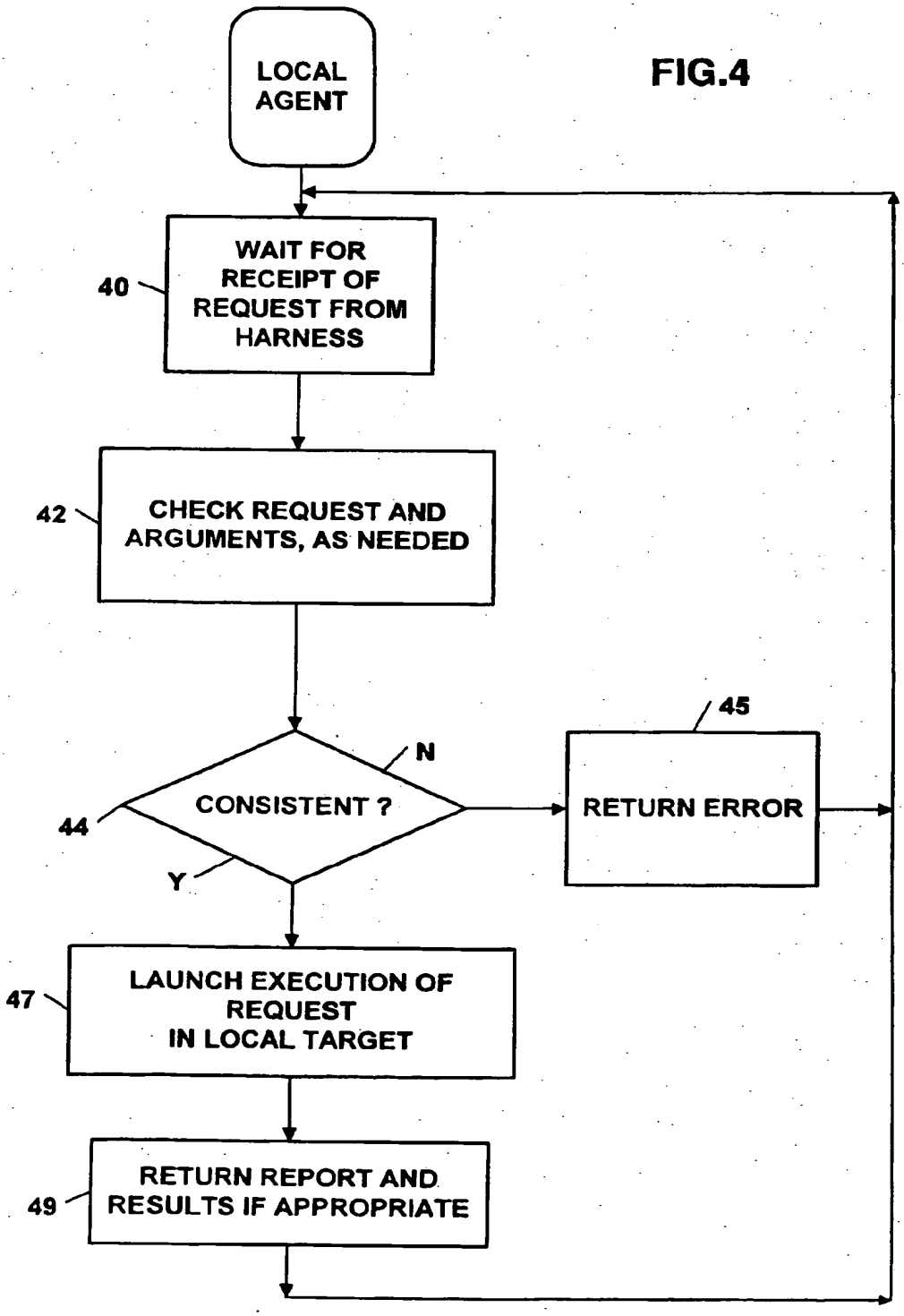

8
AGENT DEFINITIONS
(NAME,PORT,OS ARCHITECTURE)
81
82
DEFAULT SETTINGS
GENERIC TEST
SOLARIS SPECIFIC TEST CODE
841S
84
SOLARIS VERSION OF PRODUCT
842S
NT SPECIFIC TEST CODE
841N
NT VERSION OF PRODUCT
842N
GENERIC TEST CODE
850
89
RESULTS ARCHIVE

FIG.5

FIG.4



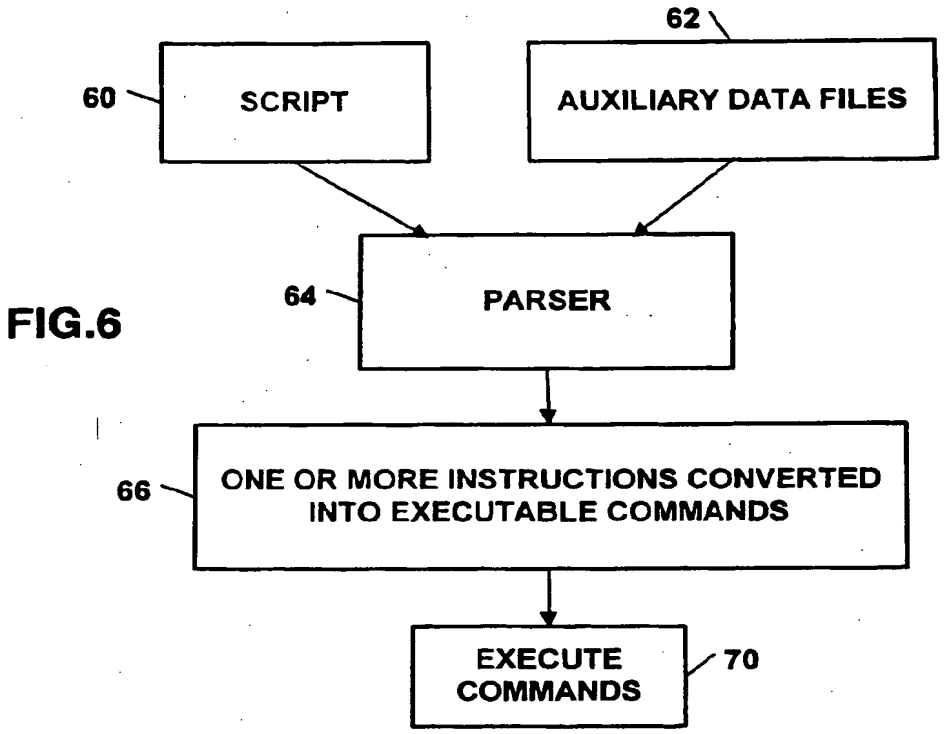

62
60
SCRIPT
AUXILIARY DATA FILES
64
PARSER
66
ONE OR MORE INSTRUCTIONS CONVERTED INTO EXECUTABLE COMMANDS
EXECUTE COMMANDS
70

FIG.6

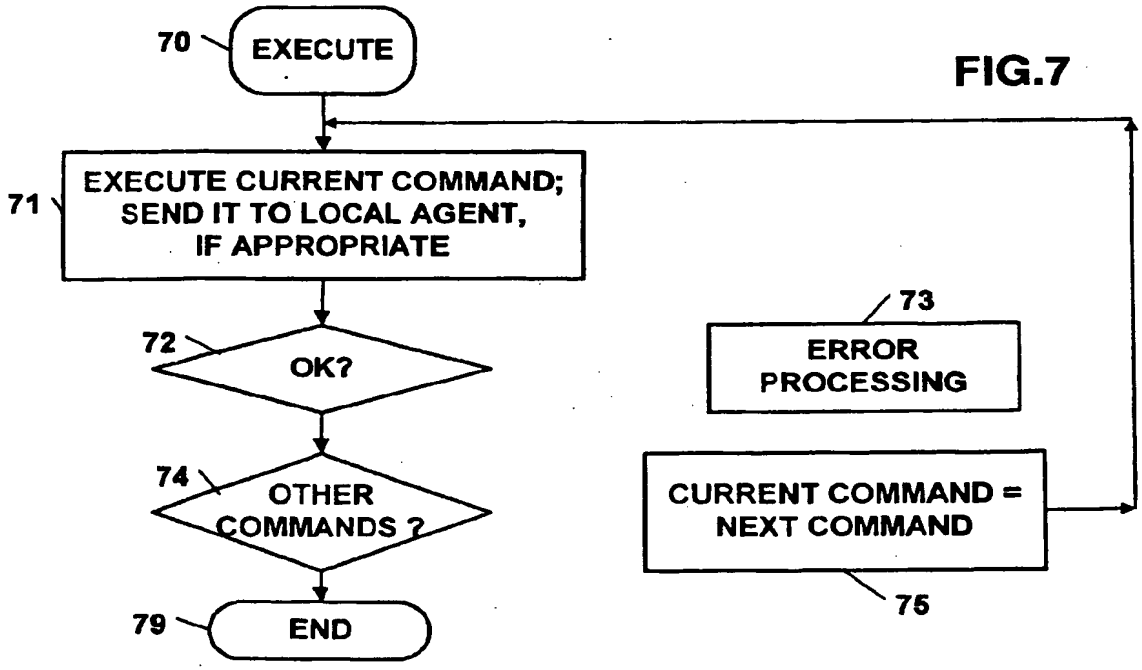

70
EXECUTE
71
EXECUTE CURRENT COMMAND; SEND IT TO LOCAL AGENT, IF APPROPRIATE
72
OK?
73
ERROR PROCESSING
74
OTHER COMMANDS ?
CURRENT COMMAND = NEXT COMMAND
75
79
END

FIG.7

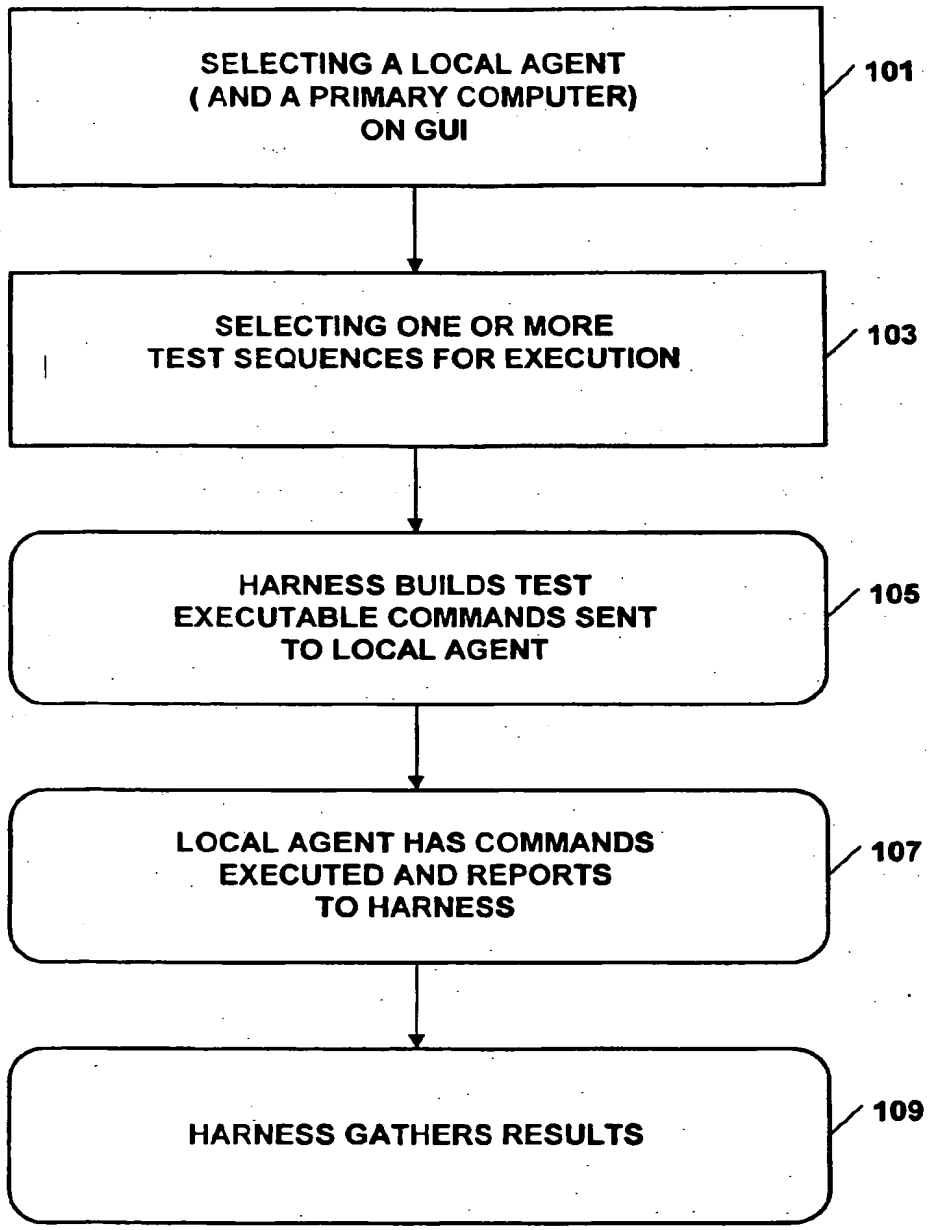


FIG.8

INTERNATIONAL SEARCH REPORT

Inter Application No
PCT/IB 00/00338

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F11/36

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, WPI Data, PAJ, IBM-TDB, INSPEC

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5 022 028 A (EDMONDS EDWARD J ET AL) 4 June 1991 (1991-06-04)	1,2,7,8, 11,13, 14,19, 20,23, 25,26
Y	abstract; figure 1 column 1, line 60 -column 2, line 27 column 4, line 9 - line 64 column 5, line 41 - line 53	3-6,10, 12, 15-18, 22,24

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
1 December 2000	08/12/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Renault, S

2

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Intern Application No
PCT/IB 00/00338

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	EP 0 964 334 A (IBM) 15 December 1999 (1999-12-15) column 1, line 9 - line 24 column 1, line 50 -column 2, line 5 column 3, line 32 -column 4, line 47 column 9, line 32 -column 10, line 52 column 11, line 35 - line 45 column 12, line 15 - line 36	3-6,10, 12, 15-18, 22,24
X	US 5 669 000 A (FLYNN SEAN LUDLOW ET AL) 16 September 1997 (1997-09-16)	1-5,7, 10,11, 13-17, 19,22, 23,25,26
A	column 4, line 39 -column 6, line 67 column 34, line 50 -column 35, line 4	8,12,20, 24
A	US 5 999 179 A (KEKIC MIODRAG M ET AL) 7 December 1999 (1999-12-07) column 14, line 8 - line 65 column 16, line 32 -column 18, line 32	1-6, 13-18, 24-26

INTERNATIONAL SEARCH REPORT

mation on patent family members

Intern Application No
PCT/IB 00/00338

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5022028 A	04-06-1991	IE 60444 B	13-07-1994
		GB 2217070 A,B	18-10-1989
		ZA 8902318 A	31-01-1990
EP 0964334 A	15-12-1999	CN 1237734 A	08-12-1999
		JP 2000029709 A	28-01-2000
US 5669000 A	16-09-1997	US 5410681 A	25-04-1995
US 5999179 A	07-12-1999	NONE	

Form PCT/ISA/210 (patent family annex) (July 1992)

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☑ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.